Juan Camilo García Saenz 2259416

Steven Girón Árcila 2259400

Jean Carlos Lerma Rojas 2259305

Informe Taller 3

1) Análisis algoritmo Divide y vencerás

Se implementa solución aplicando la estrategia de Divide y vencerás y para esta se realizaron las siguientes acciones:

A) Su lógica se basa en dividir el arreglo en dos tercios y luego llamar recursivamente al algoritmo en las primeras dos terceras partes y las últimas dos terceras partes del arreglo. Después de dividirlo, combina los resultados.

- **Divide:** En el paso de división, se comprueba si el elemento en la posición inicial (i) es mayor que el elemento en la posición final (j). Si este es el caso, intercambia estos dos elementos para asegurarse de que estén en orden ascendente. Luego, se verifica si la diferencia entre i y j es menor o igual a 1. Si esta condición es verdadera, significa que la lista está ordenada y el algoritmo termina.

- **Conquista:** Después de las comprobaciones iniciales, se calcula un valor k que representa un tercio del tamaño del subarreglo actual. Luego, se realiza una llamada recursiva a Stooge-Sort con los dos tercios iniciales (A[i] a A[j-k]) y los dos tercios finales (A[i+k] a A[j]). Esto divide el problema original en dos subproblemas más pequeños y se repite el proceso en estos subarreglos.

- **Combina:** La combinación aquí no es una combinación tradicional como en otros algoritmos de ordenamiento. Después de ordenar recursivamente las dos partes iniciales y finales del arreglo, Stooge-Sort realiza una última llamada recursiva a la primera parte del arreglo. Esto se hace para asegurar que todos los elementos estén en el lugar correcto, ya que algunos intercambios podrían haber perturbado el orden durante el proceso.

D) El algoritmo se puede representar con la siguiente relación de recurrencia: $T(n) = 3T\left(\frac{2n}{3}\right)$

$$a = 3$$
$$b = \frac{3}{2}$$
$$f(n) = O(1)$$

La complejidad del algoritmo se puede resolver con el método del maestro.
Comparando $n^{\log_b a}$ *con* $f(n)$*:*

$$n^{\log_{\frac{3}{2}}(3)} \approx n^{2.7095}$$
$$f(n) = O(1)$$

Dado que $f(n) = O(1)$ *y* $n^{\log_{\frac{3}{2}}(3)}$ *es mayor, según el caso 1 del método del maestro, la complejidad temporal del algoritmo de StoogeSort se aproxima a* $O\left(n^{2.7095}\right)$.

n	Tiempo (segundos)
10	0.003000
100	0.001000
1000	0.085000
10000	96.837000

Comparando la complejidad teórica y la práctica se puede ver que no es un algoritmo optimo. También si vemos detenidamente los datos obtenidos de la parte práctica se refleja que realmente la parte teórica se queda corta al describir la complejidad del algoritmo, ya que este a medida que incrementa los datos su eficiente disminuye.

2) Análisis algoritmo Divide y vencerás

Se realiza la solución de un algoritmo para encontrar la moda de un arreglo bajo la estrategia de divide y vencerás y para esto se realiza lo siguiente.

A) Hacemos uso del algoritmo de ordenamiento QuickSort en el cual usamos dicha estrategia y se divide en los siguientes pasos:

- **Caso base**: La función quicksort comienza con una verificación de un caso base, donde si la longitud de la lista arr es menor o igual a 1, se devuelve el mismo arreglo, ya que no hay necesidad de ordenar un arreglo de un solo elemento o vacío.
- **Divide**: Se elige un pivote, en este caso, el elemento en la mitad de la lista (arr.get(arr.size() / 2)). Se divide la lista original en tres partes:
  - ✓ **left**: Contiene elementos menores que el pivote.
  - ✓ **middle**: Contiene elementos iguales al pivote.
  - ✓ **right**: Contiene elementos mayores que el pivote.
- **Vence**: Se generan múltiples sublista en cada llamada recursiva, y esto continúa hasta que se llega a sublista de tamaño uno o cero (caso trivial), que ya se consideran ordenados.
- **Combina:** En QuickSort está implícito en la fase de construcción de la lista resultante usando “addAll”. Finalmente, se combinan los resultados ordenados de las sub left, middle, y right en una nueva lista que se devuelve como el resultado final.

Después hacemos uso de la función encontrarModa la cual recibe la lista ordenada y procedemos a:

- Creamos un mapa para almacenar la frecuencia de cada elemento de la lista.
- Calculamos la frecuencia de cada elemento iterando sobre la lista y por cada iteración al mapa de frecuencias le agregamos el elemento y si este

ya ha sido agregado le sumamos 1 a la llave del Map acumulando así la frecuencia de cada elemento.

- Posteriormente creamos una lista para almacenar las modas y una variable para conocer la máxima frecuencia y la inicializamos en 0.
- Recorremos el mapa de frecuencias y por cada iteración obtenemos la llave y el valor el cuál contiene la frecuencia de cada elemento y preguntamos si el valor es mayor a la variable de la máxima frecuencia. Si es así limpiamos la lista y agregamos la llave reasignamos el valor a la variable de la máxima frecuencia con el valor de la frecuencia actual del mapa, si esta condición no se cumple preguntamos si el valor de la frecuencia actual es igual a la máxima frecuencia y agregamos la llave actual a la lista de moda.
- Finalmente retornamos la lista de moda.

D) Para la complejidad teórica del algoritmo, tenemos 2 partes, el algoritmo de Quicksort y el método de encontrar la moda. Para el algoritmo de Quicksort tenemos que la complejidad teórica es *O* (n * log(n)), y para el algoritmo que encuentra la moda, es una complejidad lineal, es decir, *O* (n). Respecto a la complejidad práctica, tenemos los siguientes datos:

n	Tiempo (segundos)
10	0,001000
100	0,000000
1000	0,002000
10000	0,129000

Como podemos ver aun utilizando 2 algoritmos los cuales cada uno tiene su complejidad, estos logran tener un gran rendimiento y una buena optimización. Aun así en la práctica la eficiencia de los algoritmos puede varias según la implementación se aplique, por ejemplo si utiliza arreglos en lugar de listas, pero está variación no sería significativa.

3) Comparación de ejecución algoritmos
4)

Se implementan los siguientes 3 algoritmos de ordenamiento QuickSort, Insert-Sort y Merge-Sort para realizar una comparación entre ellos utilizando una tabla de entradas aleatorias de tamaño 10, 50, 100, 500, 1000, 2000, 5000 y 10000.

- QuickSort

Entrada (n)	Tiempo real (seg)	Complejidad *O* (n * log(n))	Constantes
10	0,02	10	0.002000
10	0	10	0.000000
10	0	10	0.000000
50	0	84.9485	0.000000
50	0	84.9485	0.000000
50	0	84.9485	0.000000
100	0	200	0.000000
100	0	200	0.000000
100	0	200	0.000000
500	0	1349.4850	0.000000
500	0	1349.4850	0.000000
500	0	1349.4850	0.000000
1000	0	3000	0.000000
1000	0	3000	0.000000
1000	0	3000	0.000000
2000	66020,599	6602.0599	0.001000
2000	66020,599	6602.0599	0.001000
2000	66020,599	6602.0599	0.001000
5000	369897	18494.8500	0.002000
5000	0	18494.8500	0.000000
5000	184948,5	18494.8500	0.001000
10000	80	40000	0.002000
10000	80	40000	0.002000
10000	80	40000	0.002000
		Constante	**0,0006**

- Insert-sort

Entrada (n)	Tiempo real (seg)	Complejidad O(n^2)	Constantes
10	0,2	100	0.002000
10	0	100	0.000000
10	0	100	0.000000
50	0	2500	0.000000
50	0	2500	0.000000
50	0	2500	0.000000
100	0	10000	0.000000
100	0	10000	0.000000

100	0	10000	0.000000
500	500	250000	0.002000
500	0	250000	0.000000
500	0	250000	0.000000
1000	1000	1000000	0.001000
1000	1000	1000000	0.001000
1000	1000	1000000	0.001000
2000	4000	4000000	0.001000
2000	0	4000000	0.000000
2000	4000	4000000	0.001000
5000	75000	25000000	0.003000
5000	50000	25000000	0.002000
5000	50000	25000000	0.002000
10000	800000	100000000	0.008000
10000	800000	100000000	0.008000
10000	900000	100000000	0.009000
		Constante	**0,00170833**

- Merge-Sort

Entrada (n)	**Tiempo real (seg)**	**Complejidad *O* (n * log(n))**	**Constantes**
10	0,02	10	0.002000
10	0	10	0.000000
10	0	10	0.000000
50	0	84.9485	0.000000
50	0	84.9485	0.000000
50	0	84.9485	0.000000
100	0	200	0.000000
100	0	200	0.000000
100	0	200	0.000000
500	13494,85	1349.4850	0.001000
500	0	1349.4850	0.000000
500	0	1349.4850	0.000000
1000	3	3000	0.001000
1000	3	3000	0.001000
1000	3	3000	0.001000
2000	66020,599	6602.0599	0.001000
2000	0	6602.0599	0.000000
2000	0	6602.0599	0.000000
5000	369897	18494.8500	0.002000
5000	0	18494.8500	0.000000
5000	0	18494.8500	0.000000
10000	80	40000	0.002000
10000	40	40000	0.001000
10000	40	40000	0.001000

		Constante	**0,00054167**